Sentiment Classifier

Code Presentation

Contents

Overview

Overview

Link to the Github repository: @ericdaat/sentiment-classifier

The goal of this project was to create a sentiment classifier API that could use various models and datasets.

It is written in Python and uses the following libraries:

- Flask: for the API
- Keras: for the Machine Learning Models

Folder organisation

The folder organisation is pretty standard:

- **bin**: where we store the ML trained models
- **data**: where we store the datasets
- **docs**: an auto generated documentation using python Sphinx
- **sentiment_classifier**: the actual code, containing the API the NLP parts.
- **tests**: unit tests

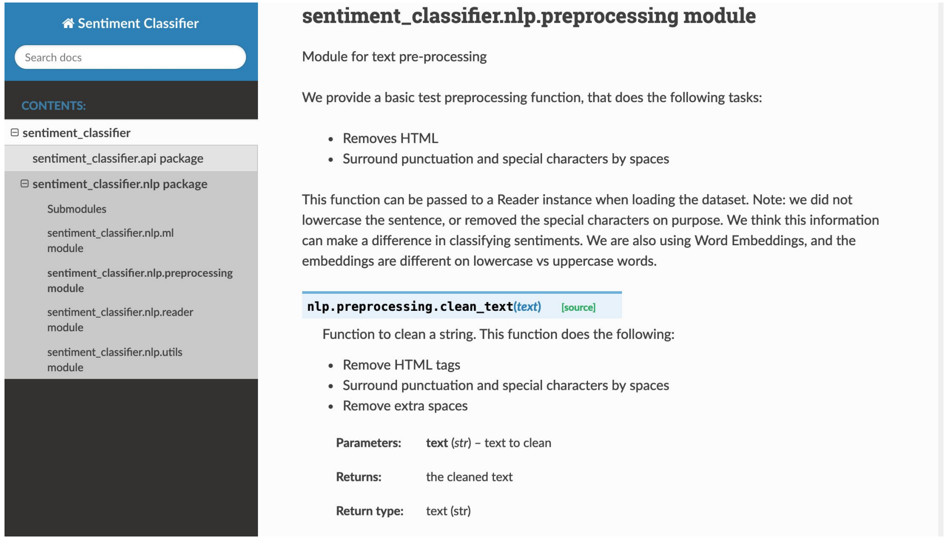
Installation & Deployment

1. Clone the repository
2. Download the IMDB dataset and place it in the data folder
3. We use pre-trained word embeddings from FastText, so you might want to download them to the data folder as well.
4. Create a virtual environment, and install the requirements from **requirements.txt** file
5. Train a model by running:
   python sentiment_classifier/scripts/train.py
6. Add “sentiment_classifier” to your PYTHONPATH, and run:
   python sentiment_classifier/api/wsgi.py

Auto-generated Documentation

The project documentation is automatically build using Sphinx, a python library.

The docstring of every function are displayed on a nice HTML page, auto generated.


Sentiment Classifier
Search docs
CONTENTS:
⊟ sentiment_classifier
sentiment_classifier.api package
⊟ sentiment_classifier.nlp package
Submodules
sentiment_classifier.nlp.ml module
sentiment_classifier.nlp.preprocessing module
sentiment_classifier.nlp.reader module
sentiment_classifier.nlp.utils module

sentiment_classifier.nlp.preprocessing module

Module for text pre-processing

We provide a basic test preprocessing function, that does the following tasks:

- Removes HTML
- Surround punctuation and special characters by spaces

This function can be passed to a Reader instance when loading the dataset. Note: we did not lowercase the sentence, or removed the special characters on purpose. We think this information can make a difference in classifying sentiments. We are also using Word Embeddings, and the embeddings are different on lowercase vs uppercase words.

`nlp.preprocessing.clean_text(text)` [source]

Function to clean a string. This function does the following:

- Remove HTML tags
- Surround punctuation and special characters by spaces
- Remove extra spaces

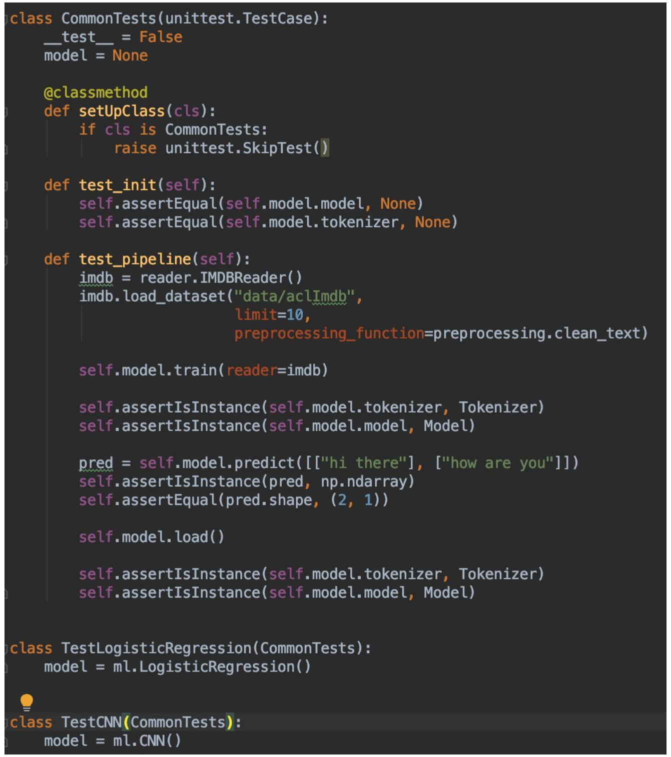
Parameters:	**text** (*str*) – text to clean
Returns:	the cleaned text
Return type:	text (str)

Unit Testing

Unit tests are written in Python, using unittest library.

They cover all the key parts of the code:

- The API
- The preprocessing
- The dataset loading
- The Machine Learning models



```python
class CommonTests(unittest.TestCase):
    __test__ = False
    model = None

    @classmethod
    def setUpClass(cls):
        if cls is CommonTests:
            raise unittest.SkipTest()

    def test_init(self):
        self.assertEqual(self.model.model, None)
        self.assertEqual(self.model.tokenizer, None)

    def test_pipeline(self):
        imdb = reader.IMDBReader()
        imdb.load_dataset("data/aclImdb",
                          limit=10,
                          preprocessing_function=preprocessing.clean_text)

        self.model.train(reader=imdb)

        self.assertIsInstance(self.model.tokenizer, Tokenizer)
        self.assertIsInstance(self.model.model, Model)

        pred = self.model.predict([["hi there"], ["how are you"]])
        self.assertIsInstance(pred, np.ndarray)
        self.assertEqual(pred.shape, (2, 1))

        self.model.load()

        self.assertIsInstance(self.model.tokenizer, Tokenizer)
        self.assertIsInstance(self.model.model, Model)


class TestLogisticRegression(CommonTests):
    model = ml.LogisticRegression()


class TestCNN(CommonTests):
    model = ml.CNN()
```

NLP Package

Dataset Reader

We are using the IMDB Large Movie Reviews dataset from Stanford AI. It provides 50,000 reviews on movies, splitted half-half in train/test and labelled as positive or negative.

We provide an abstract class **Reader** that we can subclass for each dataset. We do this to standardise the dataset loading, and make it easy to use multiple datasets in the rest of the code with a common interface.

The **IMDBReader** class implements all the code needed to load the IMDB dataset.

Text Preprocessing

We provide a basic test preprocessing function, that does the following tasks:

- Removes HTML
- Surround punctuation and special characters by spaces

This function can be passed to a **Reader** instance when loading the dataset.

Note: we did not lowercase the sentence, or removed the special characters on purpose. We think this information can make a difference in classifying sentiments. We are also using Word Embeddings, and the embeddings are different on lowercase vs uppercase words.

Utils

This module hosts various functions helpful in our NLP tasks.

So far, we only have a function responsible for loading pre-trained word embeddings.

ML

This module hosts the Machine Learning models. Every model subclasses a **Model** abstract class that has the following attributes:

- **model**: the ML model, so far built using Keras
- **tokenizer**: responsible for mapping words into indices

The **Model** class implements the following methods:

- **train**: trains the model
- **save**: saves the model weights & tokenizer
- **predict**: predicts on sentences
- **_make_training_data**: a private method that creates the train/test matrices from a **Reader** object

API Package

Application Factory

We use Flask to write the API and we are using the factory pattern to create the Flask application.

This is an elegant method that allows us to separate the code for the app creation, and register all the blueprints in one place.

The factory runs the following steps:

- Create the Flask object
- Load the ML models and attach them
- Register the **index** blueprint

Index Blueprint

This blueprint exposes two API routes:

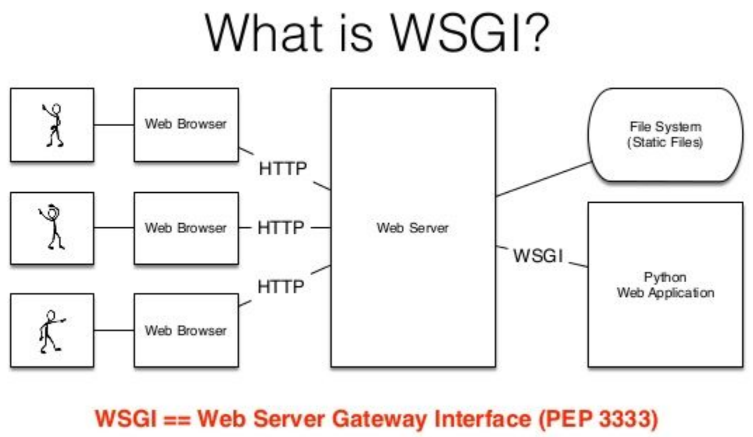
- **/api**: Allows GET method to check whether the application is up and running
- **/api/classify**: Allows POST method to compute the sentiment prediction class on a given sentence. The sentence must be passed in a json under a “text” key.

WSGI For Deployment

As mentioned in the Flask docs:
“While lightweight and easy to use, Flask’s built-in server is not suitable for production as it doesn’t scale well.”

If the app is self hosted (i.e not on Heroku or alike), we must use a WSGI server to deploy the app to production.

We chose Waitress, because it is easy to use, but uWSGI or Gunicorn are also very popular.



Machine Learning

Overview

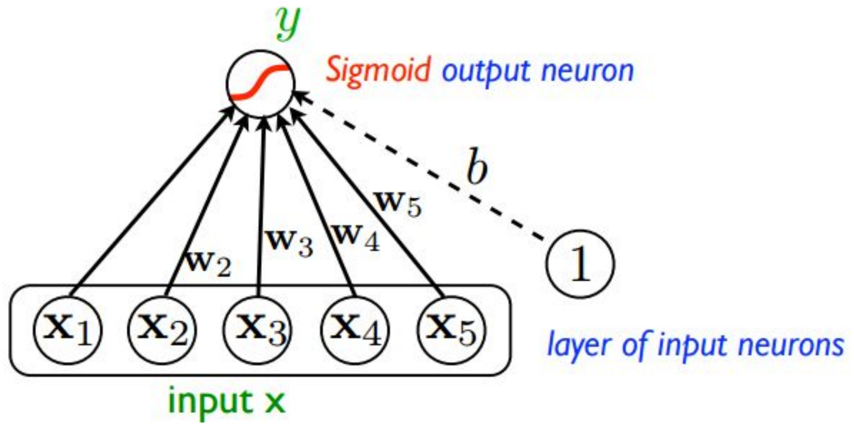
The sentiment analysis task is pretty straightforward. It is one of the first example studied when reading about Data Science & Machine Learning.

It is a binary classification task, where given a sentence we want to output 1 if the sentence is positive, and 0 if it is negative.

This problem is treated as a supervised learning approach: given a dataset of examples (texts labelled as pos/neg), we learn the model weights to minimise a loss function and generalise to unseen data.

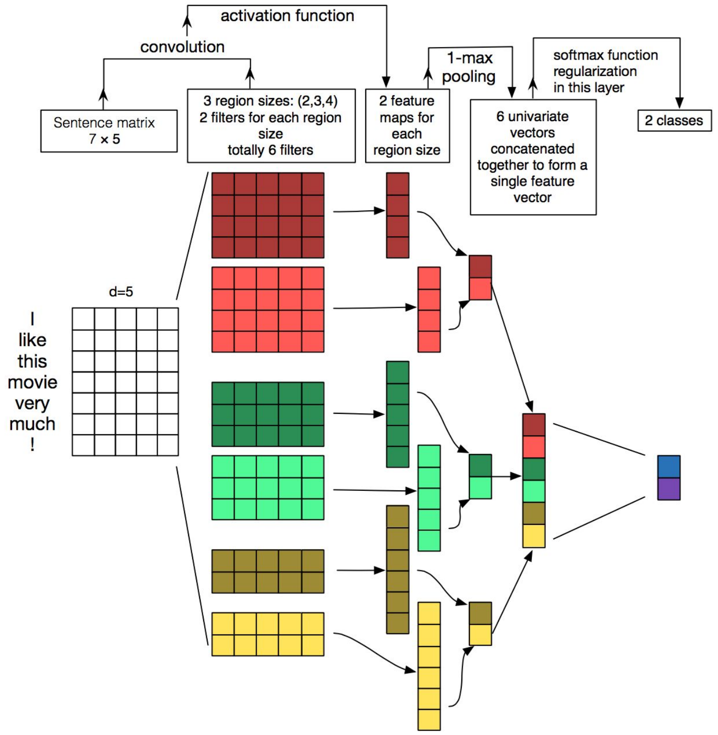
Baseline: Logistic Regression

- The words sentences are one hot encoded and used as input x
- Learn weights for every word in the vocabulary. The higher the weight, the more likely this word will make the sentence positive.
- Prediction is obtained by a weighted sum of the words from the sentence, passed into a sigmoid function to obtain a probability.



Convolutional Neural Networks for NLP

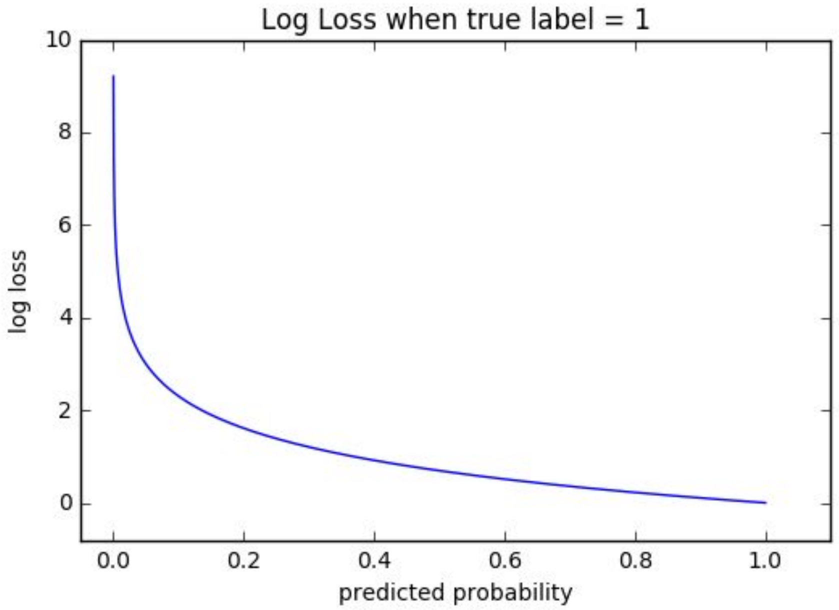
- CNN are mostly used in computer vision … but they work in NLP too!
- Map each word from a sentence to a learned embedding
- Convolve filters of different sizes on the embeddings, they are meant to learn embedding “shapes”
- Apply max pooling
- Use softmax function to obtain the classes probability



activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Cross Entropy Loss

- Used when output is a probability value between 0 and 1.
- Cross-entropy loss increases as the predicted probability diverges from the actual label.
- A perfect model would have a log loss of 0.



Results

The CNN works better than the basic Logistic Regression classifier.

However, the LR might have performed better with TFIDF, stop words, etc …

We wanted to try the CNN approach first because it is much faster to train than RNN and also because it seemed like a fun idea!

	Val Accuracy	Val Loss
Logistic Regression	0.85	0.38
CNN	**0.88**	**0.29**

Conclusion

What we have is:

- A stable-ish architecture for loading datasets and training NLP models
- A production ready API

This is a good starting point for developing more models and building a more user friendly interface around it.

Our ML models are achieving acceptable results with 0.88 mean accuracy on the validation dataset, but the State of the Art results are much higher, reaching as high as 0.95

What’s next

Next steps:

- Tune hyperparameters for existing models
- Add more NLP models
- Add more datasets
- Monitor training:
    - Store training stats in DB
    - Dataviz to show the learning curves
- Develop a Web UI

Thank you!